Manifold Dual Contouring

Scott Schaefer, Tao Ju, Joe Warren

***Abstract*— Dual Contouring is a feature-preserving iso-surfacing method that extracts crack-free surfaces from both uniform and adaptive octree grids. We present an extension of Dual Contouring that further guarantees that the mesh generated is a manifold even under adaptive simplification. Our main contribution is an octree-based, topology-preserving vertex clustering algorithm for adaptive contouring. The contoured surface generated by our method contains only manifold vertices and edges, preserves sharp features, and possesses much better adaptivity than those generated by other iso-surfacing methods under topologically safe simplification.**

***Index Terms*— iso-surfacing, contour simplification, vertex clustering, manifold**

I. Introduction

Contouring is the process of generating a piece-wise linear approximation to the zero-surface of an implicit function. Originally motivated by the need for visualizing 3D medical images, the study of contouring methods has developed into a major area in the field of graphics and visualization. A large number of these methods are designed for volumes with a uniform grid structure. For example, the Marching Cubes (MC) method [1] generates a closed, manifold triangular mesh for any signed volume. To improve the quality of the contour geometry, methods like the Extended Marching Cubes [2] have been proposed to reproduce sharp edges and corners by utilizing additional information in the volume, such as surface normals.

When the volume size is large, however, contouring on a uniform grid may generate too many polygons for visualization or further processing. To address this deficiency, Ju et al. [3] introduced the Dual contouring (DC) method for generating adaptive contours. The DC method simplifies a uniform grid into an octree structure by merging grid cells in which the underlying contour geometry is flat. DC always produces crack-free contours on any octree grid and is also capable of reproducing sharp geometry features when hermite data is available. In contrast, extending MC and its variants onto octree grids often results in cracks between the surface extracted from adjacent octree cells at different octree depths, which need to be resolved using special crack-patching strategies such as in [4], [5].

Despite being adaptive and feature-preserving, a major drawback of the DC method is that, unlike MC and many other uniform contouring methods, DC may generate non-manifold surfaces. That is, an edge on the contour may be shared by more than two polygons, and the neighborhood of a vertex may not be topologically equivalent to a disk. Non-manifold surfaces are not only less visually appealing than 2-manifolds, but also problematic for mesh processing tasks such as fairing and parameterization.

Texas A&M University, College Station, TX
Washington University, St. Louis, MO
Rice University, Houston, TX

Contributions

In this paper, we propose an extension of the Dual Contouring method that also guarantees production of manifold contours. Although there have been several variants of DC [6], [7] that introduce better topology control or even claim to produce manifold contours, non-manifold edges and vertices can still appear in the adaptive setting (see Section II). In contrast, we present theoretical proofs that our method always generates closed, 2-manifold surfaces even under adaptive simplification. Our method presents two novel additions to the original DC method:

1) A vertex clustering algorithm for contour simplification that allows multiple contour components in one octree cell. Compared to previous adaptive variants of DC [7]–[9], our method is simpler to implement and places no limit on the number of intersections between the contour and each cell edge, hence allowing less restrictive simplification.
2) A simple topology constraint in vertex clustering, which guarantees that the simplified contours are always 2-manifold. To the best of our knowledge, this is the first manifold-preserving criterion developed for octree-based vertex clustering methods.

II. Related work

In this section we briefly review the Dual Contouring method, recent extensions and variants of DC, related mesh simplification methods using vertex clustering and other approaches for topology-preserving contour simplification.

A. Dual Contouring

The Dual Contouring method, proposed by Ju et. al. [3], provides a uniform approach for extracting water-tight iso-surfaces on both uniform grids and adaptive octree grids. The algorithm creates one vertex for each grid cell that contains a sign change, and creates the surface by generating one polygon for every edge in the grid containing a sign change. Along each sign-change edge, the polygon connects the four vertices of the cells sharing that edge. DC guarantees to generate a close surface on any octree grid and can be implemented efficiently using recursive tree traversals [3].

Another advantage of DC over MC is its ability of reproducing sharp features, such as edges and corners, when Hermite data

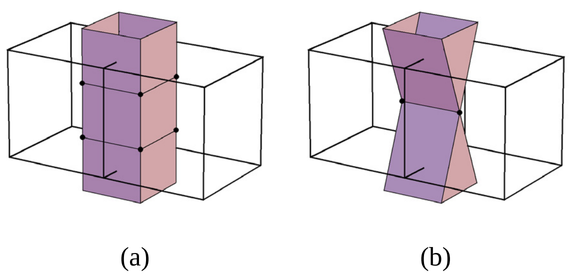
Fig. 1. Vertex clustering in two neighboring cells (a) results in a non-manifold edge on the surface (b).

is available. In Hermite representation, each grid edge that contains a sign change is associated with an intersection point between the contour and the edge as well as a normal vector of the contour at the point. Such Hermite representation can be either obtained from a closed triangular mesh [10] or directly from a implicit function. In DC, the vertex within a cell is placed so as to minimize a Quadratic Error Function [11] constructed from the Hermite data associated with the cell edges.

B. Extensions and variants of Dual Contouring

A problem of DC that has been of common interest in almost all subsequent work is the restriction of DC in maintaining no more than one contour vertex within each grid cell. To relax this restriction on a uniform grid, multiple contour components in a cell can be detected either by identifying edge-connected components of positive (or negative) cell corners [9], [12] or by utilizing the cycles in the Marching Cubes look-up table [7], [13]. In this paper, we follow the Dual Marching Cubes approach of Nielson [13] to obtain one vertex for each contour component on the uniform grid (see details in Section III).

To handle multiple vertices per cell in adaptive contouring, Zhang et al. [7] propose a vertex clustering approach for simplifying contours. Their method maintains the disconnected contour components during simplification using an *enhanced cell* representation and results in much better adaptivity than DC. However, preserving components alone is not sufficient to avoid non-manifold vertices or edges. Figure 1 shows an example in which the method of [7] would generate a non-manifold edge when vertex clustering is performed in two neighboring octree cells containing a cylinder-shaped surface. Moreover, contour simplification in [7] requires non-trivial coding-vector operations and is restricted to a maximum of 2 intersections between the contour and each cell edge. Such restriction places a bound on the maximum number of contour patches that an octree cell may contain, resulting in limited simplification of complex contours (see an example in Section VI). A similar restriction is also found in the method of Varadhan et al. [9], which creates an adaptive grid using octree refinement guided by feature detection.

Instead of contour simplification, the method of Ashida and Badla [6] extracts contours directly from octrees with adaptive resolution. Their method identifies cycles of contour faces intersecting each octree cell and creates one vertex for each cycle. Despite their claim of a manifold contour, non-manifold contour edges may still appear between two neighboring cells, such as in the cylinder example of Figure 1 (b). Moreover, cycle-identification is a time-consuming process. In a completely different approach, Schaefer et al. [8] extract contours by performing MC on a hexahedral grid dual to the octree grid, which is generated by extending DC to volumetric functions. The resulting surface is guaranteed to be a 2-manifold, but is very expensive to compute.

C. Vertex clustering

Contour simplification in DC and its variants is closely related to vertex clustering methods for simplifying polygonal meshes. These methods group vertices based on spatial or geometric proximity and compute one representative vertex for all vertices in a same group. Vertex grouping often utilizes some type of spatial partitioning structure, such as uniform cubic grids [14], floating cells [15], octree grids [16] and BSP trees [17]. As in DC, Quadratic Error Functions can be used for accurate placement of representative vertices [18], [19]. However, little work has been done in controlling topology during vertex clustering. Brodsky and Watson [20] perform a topology check that partitions a group of vertices if the group contains disjoint components. Similarly, Kanaya et al. [21] compute one representative vertex for each connected component in each vertex group to preserve disjoint portions of a mesh. To date, there has been no report of any octree-based vertex clustering method that preserves the manifoldness or genus of the surface.

D. Topology-preserving contour simplification

Besides the octree-based vertex-clustering approach in DC and its variants, there are several other contour simplification methods, some of which preserve surface topology during simplification. Lewiner et al. [22] presented a iso-surface compression method on a simplicial (e.g., triangular or tetrahedral) grid via simplification operators, known as “welds”, that are applied to the grid. The compression preserves iso-surface topology and manifoldness by checking the Euler characteristic of the surface portion affected by each weld. However, such test is computationally expensive as the fine iso-surface has to be computed locally prior to each weld.

Another approach for contour simplification is to contour a uniform grid first and then simplify the resulting iso-surface using a main-stream mesh simplification technique such as [23] or [11]. In contrast to DC and its variants, which apply grid simplification first and then contour, this second approach can be much more time and space consuming due to the need to generate and store a fine polygonal iso-surface prior to mesh simplification.

To reduce the high cost of the contour-and-simplify approach, Attali et al. [24] proposed a hybrid approach where a fine iso-surface is formed and immediately simplified as each slice of the grid is processed. The iso-surface is contoured using MC and simplification is based on edge contractions on the polygonal surface. By enforcing the “link conditions” proposed by Dey et al. [25] during simplification, the simplified surface is always manifold and preserves the topology of the original iso-surface. While Attali’s method avoids storing the entire, fine-level iso-surface, the speed of the method remains slow since this fine surface stills needs to be generated and then simplified. As we will see, our topology-preserving modification to DC simplifies an iso-surface in much less time since no polygon is generated until after the grid is simplified.

III. Contouring on a uniform grid

We start by describing a simple modification, first proposed by Nielson [13], to the original DC algorithm [3]. One of the limitations of DC is that it allows no more than one vertex within each grid cell. On a uniform grid, DC leads to non-manifold vertices and edges for all of the ambiguous sign configurations in the original Marching Cubes algorithm [1].

To combat this effect, Nielson’s modification allows multiple vertices to be placed in a single cell. In particular, Nielson associates one vertex with each cycle of a modified Marching Cubes table [26]. Since each cycle consists of a list of edges on the cubic cell, each vertex is associated with a set of edges and each edge is associated with exactly one vertex. To create polygons, the algorithm constructs one polygon connecting the vertices associated with that edge in the four adjacent cells. This algorithm creates a quadrilateral surface that is the dual of the surface created using Marching Cubes (and was therefore given the name “Dual Marching Cubes”). Furthermore, this surface is always a manifold because the original Marching Cubes algorithm always constructs a manifold and the dual preserves the topology of the surface.

One of the advantages of DC over a traditional contouring method, such as Marching Cubes, is its capability of reproducing sharp features in the presence of Hermite data. To incorporate Hermite data into Nielson’s Dual Marching Cubes algorithm, we simply construct a Quadratic Error Function (QEF) [11] for each vertex using the Hermite data on the edges associated with that vertex. We place this vertex at the location that minimizes that error function.

Figure 2 shows a comparison in 2D of the different methods. Marching Cubes always produces a manifold but does not reproduce sharp features. Dual Contouring reproduces sharp features but the topology may be non-manifold in some configurations. The Hermite extension to Dual Marching Cubes always produces a topological manifold and can reproduce sharp features as well.

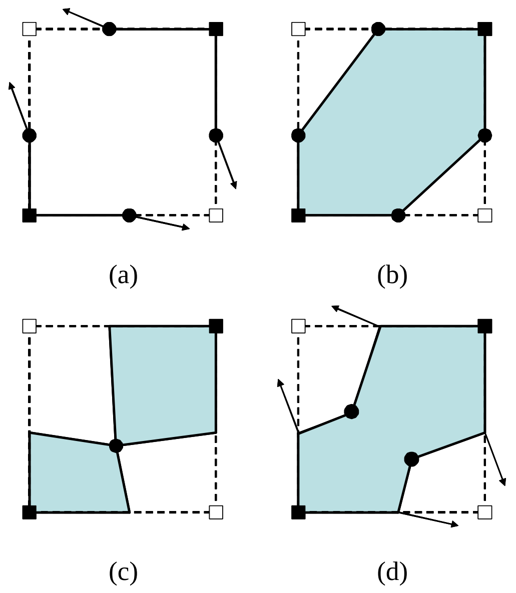

Fig. 2. Comparison of contouring with hermite data. Cell with hermite data on edges (a), Marching Cubes (b), Dual Contouring (c) and Hermite Dual Marching Cubes (d).

IV. Adaptive contouring

In the previous section, we considered constructing manifold iso-surfaces from uniform grids that preserves sharp features. However, for models with relatively flat regions, the uniform contouring algorithm produces a large number of polygons covering these flat regions. Ideally the contouring algorithm would extract a surface where the number of polygons adapts to the local properties of the surface (i.e; fewer polygons in flat regions).

DC provides such an algorithm to construct multi-resolution iso-surfaces. The method essentially performs vertex clustering where the vertices of the child cells in the octree collapse to a single vertex in a topologically safe manner. However, since only one vertex was allowed per-cell in DC, the collapse was very restrictive. Here we develop a new contour simplification method via octree-based vertex clustering, which allows for an arbitrary number of vertices per cell. Furthermore, we describe a polygon generation algorithm for constructing surfaces from these adaptively clustered vertices.

A. Vertex clustering

Given an error threshold, the vertex clustering phase creates a vertex tree starting with the vertices at the finest level of the octree. Each vertex contains a parent pointer as well as the QEF associated with this vertex and the value of the QEF evaluated at this vertex (i.e; the error associated with this vertex). Furthermore, a vertex is marked as being *collapsible* if the error associated with the vertex is less than our given threshold. Initially, we flag all vertices as collapsible and set their parent indices to NULL.

When simplifying the octree, we only cluster vertices together that are topologically connected on the surface. Note that this approach is similar to Zhang et al. [7], but it is not sufficient to guarantee that we maintain the manifold properties of the surface under simplification (which will be addressed in Section V).

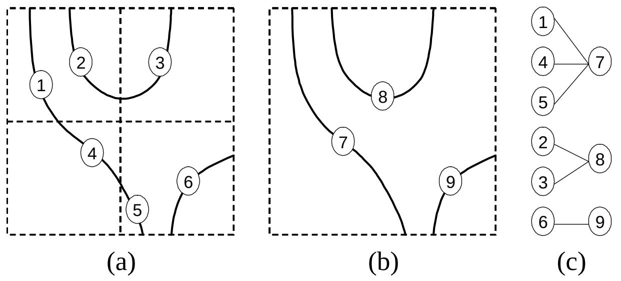

Fig. 3. Surface and vertices before clustering (a) and after clustering (b), and the vertex tree generated by this collapse (c).

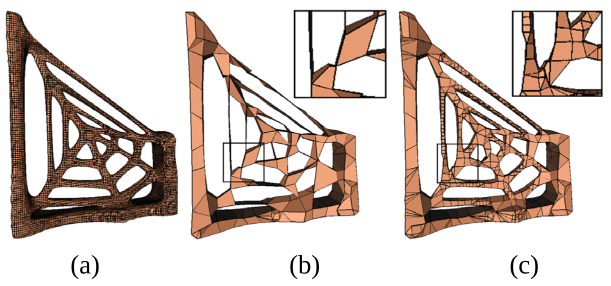

Fig. 4. A spider web contoured at a uniform resolution (a), simplified using adaptive contouring without (b) and with (c) topology constraint. The magnified region highlights some of the non-manifold regions.

Our method traverses the octree cells from the bottom up. For each octree cell that is not a leaf, we consider its eight children. These children have twelve faces that are internal to their parent cell (4 for each of the Euclidean axes). We cluster together vertices at the root of the vertex tree that are topologically connected by edges dual to the twelve internal faces. The recursive octree traversal algorithm in Ju et al. [3] provides an efficient technique for finding all of these edges. For each group, we cluster the vertices together by combining their QEF's and minimizing the error function to find the new vertex location as well as the error associated with this new vertex. If the error for this vertex is less than the threshold, we mark the new vertex as collapsible.

Figure 3 shows a $2D$ illustration of this algorithm. Here a quadtree has four children and we cluster vertices together that are connected by edges through the four internal grid edges. The vertex trees (see Figure 3 (c)) are maintained independent of the actual octree. If we compare our approach with Zhang et al. [7], which builds a vertex tree by merging "coding vectors" associated with vertices, we can maintain similar topological connectivity without resorting to complex coding for each vertex inside of the cell. Also, we can handle an arbitrary number of intersections per edge whereas other methods [7]–[9] restrict the number of intersections to two.

Note that so far we permit a surface of arbitrarily complex topology to be clustered into a single vertex, which may yield non-manifold topology after clustering. We will resolve this deficiency in Section V by introducing an additional topology criterion for collapsible vertices, which will restrict clustering to surfaces with simple topology (e.g., a sheet) within each cell.

B. Polygonalization

After the vertex clustering stage, we construct polygons that connect these vertices together. The vertices included in the output mesh will be those vertices marked as being collapsible that do not have any collapsible ancestors in the tree.

To construct polygons, we follow the uniform contouring algorithm and create a polygon connecting the vertices associated with each edge that exhibits a sign change. For each of those vertices, we follow the parent pointers up the vertex tree to find the last vertex marked as being collapsible. If the resulting polygon collapses to an edge or a vertex, then we discard that polygon and continue.

To enumerate these edges, we use the recursive algorithm detailed by Ju et al. [3], which traverses the octree and collects the octree cells adjacent to each of the edges. Their algorithm involves three types of functions `cellProc`, `faceProc` and `edgeProc` that enumerate the cells, faces and edges of the octree along with their adjacent octree cells. For further details, we refer the reader to their paper.

One disadvantage of the above algorithm is that it requires a traversal of the entire octree even after vertex clustering has collapsed vertices. To optimize this algorithm, we mark a cell during the clustering algorithm as "collapsed" if all clustered vertices created in that cell satisfy the collapsible criterion and all of the children of that cell are either leaf cells or marked as collapsed. If a cell is collapsed, then none of the children cell in this octree cell create any polygons and we can truncate the octree traversal (`cellProc`) when it encounters a collapsed cell. Furthermore, we can truncate the `faceProc` traversal on a face if both cells sharing the face are collapsed cells, because no polygons corresponding to the shared face will be generated.

Figure 4 demonstrates the result of adaptive simplification. The left-hand side shows a spider web created without simplification. Performing vertex clustering to a predefined error tolerance yields the next image. Notice that, just because we only cluster vertices together that are topologically connected, we do not necessarily maintain the manifold properties of the surface. In particular, many of the threads in the web have collapsed to single polygons or non-manifold edges. The manifold criterion in the next section provides a method for detecting these unsafe collapses and marking those vertices appropriately.

V. Manifold vertex clustering

In the previous section, a vertex is marked collapsible during clustering if its associated QEF error is less than a given threshold. In this section we require a collapsible vertex to satisfy an additional topology criterion, so that the simplified contour is a 2-manifold, that is, every contour edge is shared by two polygons and every vertex is surrounded by a disk-like neighborhood. We will first present the criterion, and then

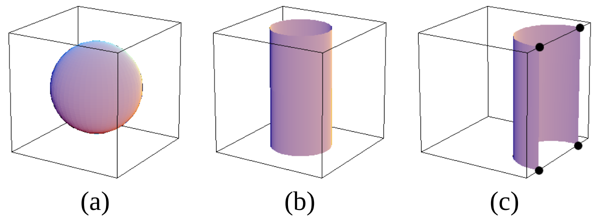

Fig. 5. Examples of surface portions that do not satisfy the manifold clustering criterion. (a) A sphere has Euler characteristic 2. (b) A cylinder has Euler characteristic 0. (c) A half-cylinder has 4 edge intersections on a face of the cell.

we explain how the quantities used in the criterion can be efficiently obtained during clustering.

A. Manifold criterion

Given a vertex v on a simplified contour, we define C_v as the octree cell in which v was created by clustering, and S_v as the set of all polygons on the *un-simplified* surface (extracted at the finest level of the octree) incident on vertices that are clustered to v. Intuitively, S_v is collapsed to the 1-ring neighborhood of v on the simplified contour. Since we want the neighborhood of v to be equivalent to a disk, we would like S_v to have a single, connected boundary. An important quantity that will help us establish this property is the Euler characteristic $\chi(S_v)$, which is defined by counting the numbers of edges E, faces F and vertices V of S_v:

$$\chi(S_v) = V(S_v) - E(S_v) + F(S_v) \qquad (1)$$

Now we present our main result (see proof in Appendix I):

Proposition 1: The adaptive contouring algorithm in Section IV generates a closed 2-manifold if, for every vertex v on the contour,

1) $\chi(S_v) = 1$; and
2) The number of intersections of S_v with the four edges of each face of C_v is either 0 or 2.

Intuitively, condition (1) allows only portions of the surface equivalent to an open disk to be collapsed to a vertex, and condition (2) further ensures that such collapsing only creates edges contained by exactly 2 polygons. As counter-examples, Figure 5 illustrates several cases of S_v that do not satisfy the manifold criterion. The first two surfaces do not have an Euler characteristic of 1 and clustering will remove a surface component or result in a non-manifold vertex. In the third example, which violates condition (2), clustering may introduce a non-manifold edge shared by four polygons, as shown in Figure 1, if the other half of the cylinder is clustered to a vertex in the neighboring cell.

Note that the two conditions in Proposition 1 are sufficient, but not necessary. The reason that we consider this particular set of conditions is that they can be efficiently verified during the bottom-up octree collapse (see next sub-section). In addition, the two conditions apply to each clustered vertex independently. Since a cell may contain multiple clustered vertices corresponding to multiple disjoint components, condition (2) places no limits on the number of intersections between the contour and each cell edge. Furthermore, since condition (1) implies that the genus of surface S_v is zero (see Appendix I), vertex clustering not only preserves the manifold properties of the surface, but also the genus of the un-simplified surface.

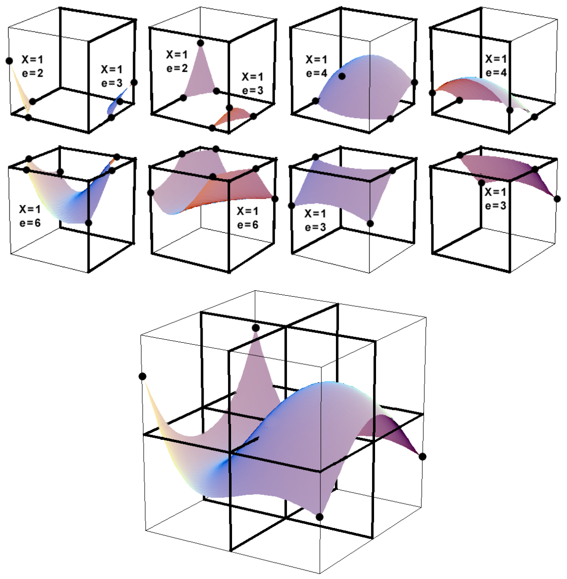

Fig. 6. Computing Euler characteristic of S_v (bottom) that is the union of several components S_{v_k} (top). Edge intersections are drawn as dots, X denotes the Euler characteristic of each S_{v_k}, and e denotes the number of intersections between S_{v_k} and the thickened edges.

B. Computing edge intersections and Euler characteristic

The manifold criterion presented in Proposition 1 requires us to compute the Euler characteristic $\chi(S_v)$ and the number of intersection of S_v on the 12 edges of C_v. However, directly computing $\chi(S_v)$ using the definition in equation (1) requires the knowledge of points, edges and polygons in each S_v. As clustering proceeds, the size of S_v becomes larger for v at higher level of the vertex tree, and such computation becomes more time-consuming. Delfinado and Edelsbrunner [27] first introduced an incremental algorithm that can be used to compute Euler characteristic of a growing triangular surface that expands by adding one triangle at a time. Based on the recursive nature of our vertex clustering, we present a simple, recursive algorithm for computing both $\chi(S_v)$ and edge intersection numbers from previously clustered vertices.

During vertex clustering, we compute 13 numbers for each vertex v. These numbers include $\chi(S_v)$ and $e_i(S_v)$ for $i = 1,\ldots,12$, which is the number of intersections of S_v on the i^{th} edge of C_v. Starting with the base case of a leaf cell, these quantities are easy to compute. In this configuration, S_v consists of a single vertex v connected to polygons dual to the edges associated with v. Therefore, $e_i(S_v)$ is 1 for each edge associated with v and 0 otherwise. Furthermore, $\chi(S_v) = 1$, which can be trivially verified using equation 1.

To create a recursive algorithm for computing $e_i(S_v)$ and $\chi(S_v)$

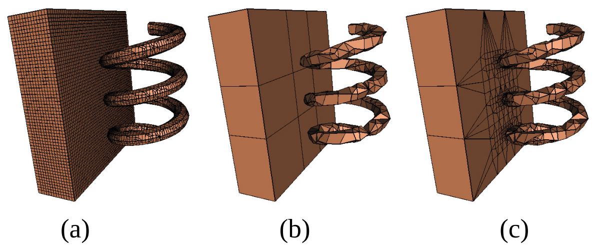

Fig. 7. (a): A spring model contoured on a uniform grid. (b): A model simplified using our method allows each surface to be simplified independently. (c): Dual Contouring restricts simplification even for separate surfaces.

for a clustered vertex v, we observe that the surface S_v is the union of surfaces S_{v_k} where v_k are the vertices clustered together to form v from the child cells of C_v. To compute $e_i(S_v)$, we simply sum the number of intersections of each S_{v_k} along edges of C_v, shown as thin lines in Figure 6 (bottom). We can compute $\chi(S_v)$ in an equally efficient manner using an inductive formula that relates $\chi(S_v)$ to $\chi(S_{v_k})$ (see proof in Appendix II):

$$\chi(S_v) = \sum_k \chi(S_{v_k}) - \frac{\sum_k e(S_{v_k})}{4} \tag{2}$$

where $e(S_{v_k})$ denotes the sum of the number of intersections of S_{v_k} along the internal edges of C_v, shown as thickened lines in Figure 6 (bottom).

Figure 6 shows an example where S_v (bottom) is built from ten surfaces S_{v_k} (top). For each child cell we display the quantities $\chi(S_{v_k})$ and $e(S_{v_k})$ for each S_{v_k} in Figure 6 (top). Observe that $\sum_k \chi(S_{v_k}) = 10$ and $\sum_k e(S_{v_k}) = 36$, and hence $\chi(S_v) = 1$ by equation 2, which is the correct Euler characteristic of the disk-like surface S_v.

To integrate the topology constraint into the adaptive contouring algorithm in the previous section, we require that a vertex v is collapsible if the associated QEF error is below the given threshold *and* if $e_i(S_v)$ and $\chi(S_v)$ satisfy the two conditions in Proposition 1. Figure 4 (c) shows the result of adaptive contouring with topology constraint, which preserves all the threads of the spider-web with manifold vertices and edges.

VI. Results

Compared with other contour simplification algorithms such as the original Dual Contouring method or the extended Dual Contouring method by Zhang et al. [7], our algorithm is much less restrictive in the types of simplifications allowed. First, multiple contour components within a same octree cell simplify in an independent manner, hence allowing flat regions to maximally collapse even if in the vicinity of other geometry (see Figure 7). Second, unlike [7], our method puts no restriction on the number of contour intersections on each octree cell edge, as our vertex tree is separate from the octree. This allows us to simplify multiple layers of thin geometry. A 2D example is shown in Figure 8, where our proposed method is capable of simplifying nearby layers of contours much better than both DC and Extended Dual Contouring [7].

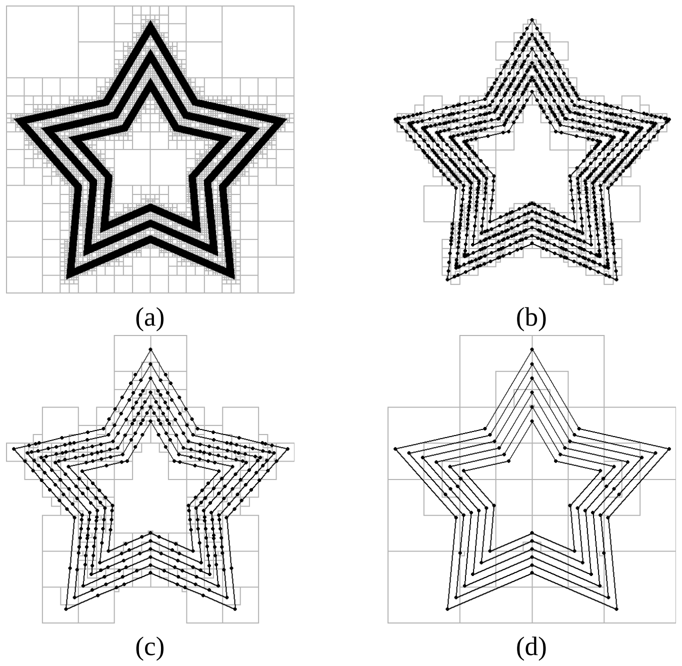

Fig. 8. Comparison between Dual Contouring (b), Extended Dual Contouring [7] (c), and the proposed Manifold Dual Contouring (d) in simplifying a 2D contour (a). Vertices and edges on the contour are drawn as round dots and lines, and octree cells in which the clustered vertices are formed are shown.

Furthermore, most contour simplification algorithms [3], [7] stop simplifying surface components as soon as an unsafe simplification is encountered, which limits the amount of simplification possible. In contrast, our manifold criterion may be able to determine that a safe simplification occurs later in vertex clustering even if unsafe collapses occurred previously. This method allows for extreme simplifications where even very dense models such as Figure 9 collapse to extremely simple shapes.

Figures 10 and 11 show two other complex scanned models that have been simplified using our method by varying the error threshold (the hermite volume representations of each model were obtained using the PolyMender tool [10]). Each model is topologically equivalent to (i.e, having the same genus as) the original and does not contain any non-manifold edges or vertices.

One attractive feature of our vertex clustering algorithm is that, once the vertex tree is constructed, simplified polygons can be generated efficiently off the vertex tree given different user-specified QEF error thresholds. This is done simply by revising the "collapsible" tag of each clustered vertex according to the new error threshold, and there is no need to rebuild the vertex tree. In contrast, methods that first build the fine-level contour followed by mesh simplification (such as [24]) would need to re-run the entire simplification process when the error threshold is changed. Such feature of our algorithm could be useful, for example, in realtime navigation of a complex volume. In these applications, the QEF error thresholds are higher in octree cells that are further away from the viewer's location, resulting in more detailed geometry in the viewer's vicinity and coarser polygons at distances. Figure 12 shows

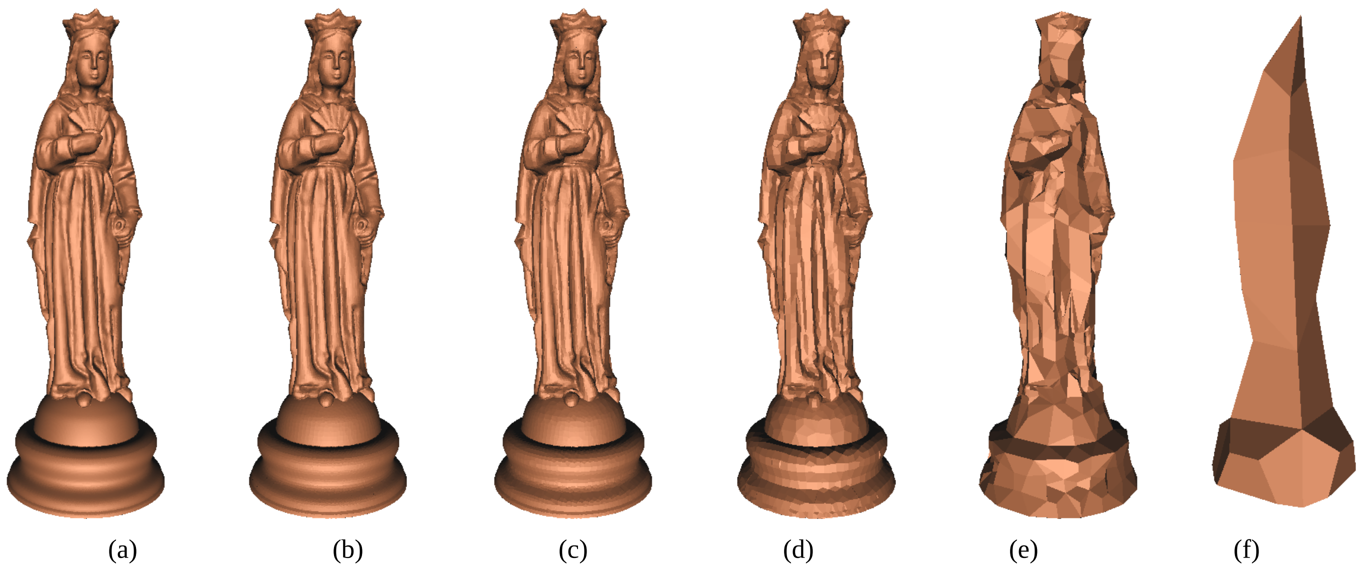

Fig. 9. Adaptive generation of iso-surfaces. Each model is guaranteed to be topologically a manifold: the iso-surface extracted on a uniform grid (a) contains 476184 polygons, and adaptive simplification generates models (b-f) with 142570, 62134, 14335, 2738 and 78 polygons.

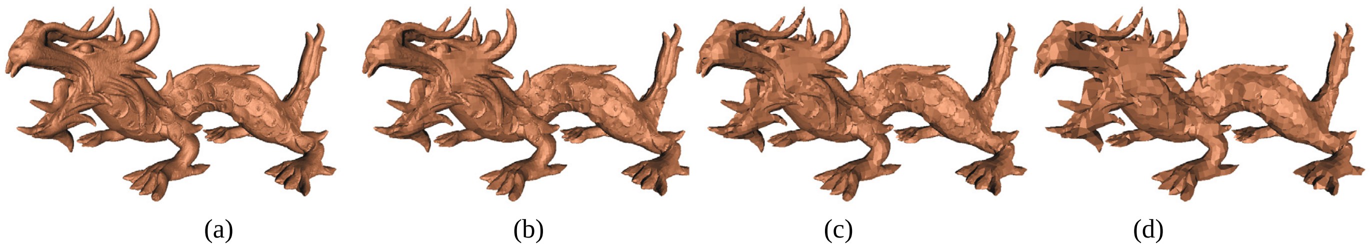

Fig. 10. Adaptive contouring of a dragon. The original contour on the uniform grid (a) contains 611476 polygons, and the following adaptive contours (b-d) contain 74770, 39800 and 20580 polygons while maintaining the manifoldness and genus of the original surface.

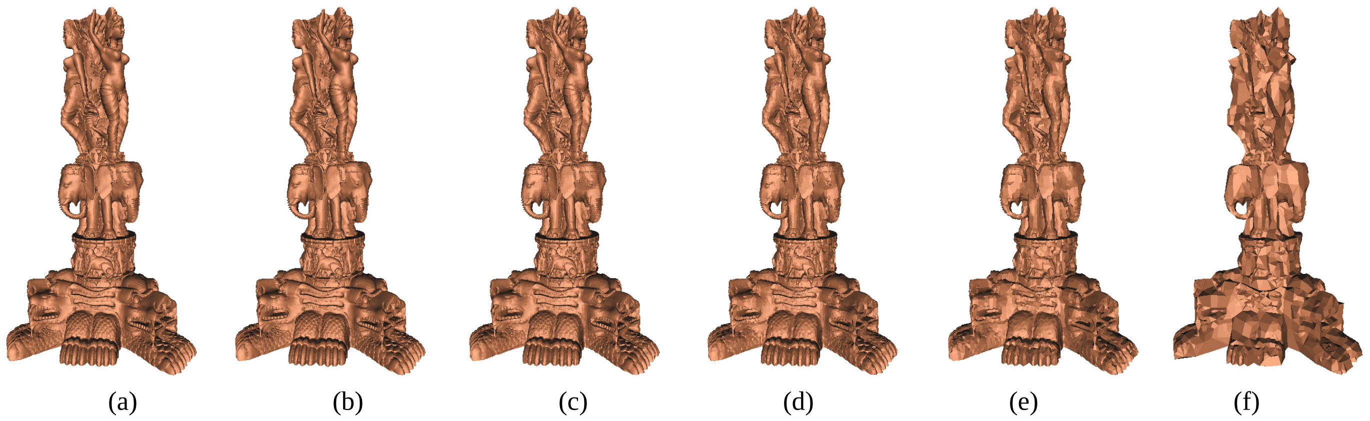

Fig. 11. Adaptive contouring of a statue with fairly complex geometry. Each adaptive contour in (b-f) is a manifold and topologically equivalent to the original contour on the uniform grid (a). The surfaces have 878368, 550984, 283948, 138516, 74964 and 30002 polygons respectively.

third-person views of the Queen model visualized with respect to different viewer's locations (marked as blue dots). After initial vertex clustering, computing each view involves only polygon generation, and each simplified surface is guaranteed to be a manifold that preserves the topology of the original iso-surface.

Finally, Table I contains timing results for our algorithm on a 3 GHz Pentium computer with 2 GB main memory. The time taken to simplify a shape is dominated by the vertex clustering phase. In particular, we compare in Table I the performance between clustering without the manifold criterion (i.e, involving only summing and minimizing QEFs) and clustering while checking the manifold criterion for each clustered vertex. Observe that, using the fast recursive algorithm presented in Section V(B), the extra computation time for enforcing manifold criterion is negligible comparing to that for QEF operations. If we compare Table I with the performance of Attali's method reported in [24], our method runs approximately an order of magnitude faster on iso-surfaces with a similar complexity at a same grid resolution.

	Octree Depth	Base Polygons	Clustering Time (sec) Without Manifold Criterion	Clustering Time (sec) With Manifold Criterion	Polygon Generation Time (sec)	Simplified Polygons
Fig 7	6	28740	0.254	0.259	0.060	1042
Fig 4	7	44784	0.459	0.465	0.097	3672
Fig 9	9	476184	5.58	5.76	1.12	78
Fig 10	9	611476	6.65	6.71	1.42	9944
Fig 11	9	878368	10.89	10.99	2.01	30002

TABLE I

SIMPLIFICATION TIME IN SECONDS FOR THE VARIOUS STAGES (CLUSTERING AND POLYGON GENERATION), COMPARING CLUSTERING WITH AND WITHOUT THE MANIFOLD CRITERION.

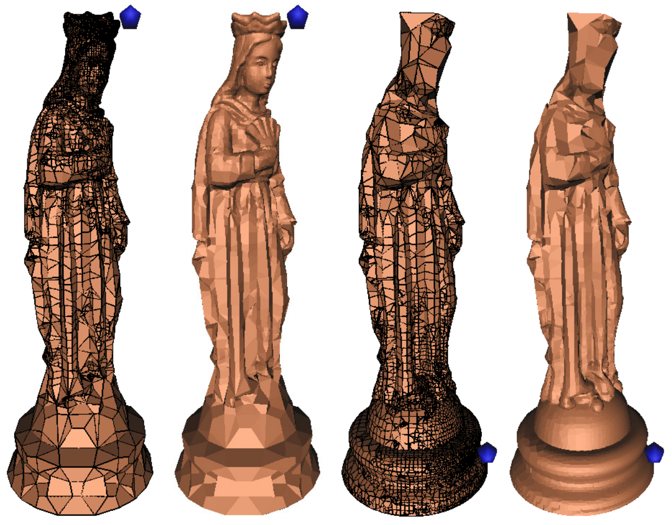

Fig. 12. Simplifying the contour based on two different viewer's locations (marked as blue dots), near the head (a,b) and near the base (c,d), shown with polygon edges (a,c) and without edges (b,d). Note that the surface further away from the viewpoint is simplified more.

VII. CONCLUSIONS AND FUTURE WORK

We have presented an extension to Dual Contouring that preserves sharp features and always constructs a manifold surface. Furthermore, we developed a simple criterion for vertex clustering in an octree that is guaranteed to preserve the genus of the original surface and always produce a 2-manifold without any non-manifold vertices or edges.

Though the surfaces we produce are topologically manifold, they may still contain intersecting polygons. For example, in our Hermite extension to Nielson's Dual Marching Cubes algorithm, we may place multiple vertices inside of a cell. It is possible that the Hermite data along the cell edges causes the vertices to be positioned such that the surfaces intersect within the cell. Note that intersecting polygons may arise even when a single vertex is placed inside a cell, as observed in [28]. As a result, the original DC algorithm as well as its variants are all subject to such geometric errors.

A naive approach for detecting intersecting polygons generated by DC-like methods involves time-consuming neighbor-finding on the octree as each polygon spans multiple octree cells. Instead, [28] presented an efficient, intersection-free modification to the original DC method by devising a set of simple geometric tests to identify potentially intersecting polygons, which are then tessellated into smaller, non-intersecting triangles. While the method of [28] is restricted to single vertex per octree cell, in the future we would like to extend such method and explore criteria for placing multiple vertices within a cell that both reproduces sharp-features and avoids intersections even under adaptive simplification.

Given that the simplified iso-surface using our approach preserves the topology of the original model, an interesting direction that worth investigating is how our method can be combined with topology-repair algorithms for large meshes, and in particular, the grid-based methods such as [29], [30]. We anticipate that a geometrically simplified yet topologically equivalent surface would greatly accelerate the process of locating topological errors in these methods.

Acknowledgements

We would like to thank the Stanford 3D Scanning Repository for the Dragon and Thai Statue models, Cindy Grimm for the Spider-web model, and VanDuzan for the Queen model. We would also like to thank the anonymous reviewers for their comments and suggestions.

REFERENCES

[1] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," in *Computer Graphics (Proceedings of SIGGRAPH 87)*, vol. 21, no. 4, Anaheim, California, July 1987, pp. 163–169.

[2] L. P. Kobbelt, M. Botsch, U. Schwanecke, and H.-P. Seidel, "Feature-sensitive surface extraction from volume data," in *Proceedings of SIGGRAPH 2001*, ser. Computer Graphics Proceedings, Annual Conference Series. ACM Press / ACM SIGGRAPH, August 2001, pp. 57–66.

[3] T. Ju, F. Losasso, S. Schaefer, and J. Warren, "Dual contouring of hermite data," *ACM Transactions on Graphics*, vol. 21, no. 3, pp. 339–346, July 2002, iSSN 0730-0301 (Proceedings of ACM SIGGRAPH 2002).

[4] R. Shekhar, E. Fayyad, R. Yagel, and J. F. Cornhill, "Octree-based decimation of marching cubes surfaces," in *VIS '96: Proceedings of the 7th conference on Visualization '96*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1996, pp. 335–ff.

[5] R. Westermann, L. Kobbelt, and T. Ertl, "Real-time exploration of regular volume data by adaptive reconstruction of isosurfaces," *The Visual Computer*, vol. 15, no. 2, pp. 100–111, 1999. [Online]. Available: citeseer.ist.psu.edu/westermann99realtime.html

[6] K. Ashida and N. I. Badler, "Feature preserving manifold mesh from an octree." in *Symposium on Solid Modeling and Applications*, 2003, pp. 292–297.

[7] N. Zhang, W. Hong, and A. Kaufman, "Dual contouring with topology-preserving simplification using enhanced cell representation," in *VIS '04: Proceedings of the conference on Visualization '04*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 505–512.

[8] S. Schaefer and J. Warren, "Dual marching cubes: Primal contouring of dual grids," in *PG '04: Proceedings of the Computer Graphics and Applications, 12th Pacific Conference on (PG'04)*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 70–76.

[9] G. Varadhan, S. Krishnan, Y. Kim, and D. Manocha, "Feature-sensitive subdivision and iso-surface reconstruction," in *IEEE Visualization 2003*. IEEE, 2003, pp. 99–106.

[10] T. Ju, "Robust repair of polygonal models," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 888–895, 2004.

[11] M. Garland and P. S. Heckbert, "Surface simplification using quadric error metrics," in *Proceedings of SIGGRAPH 97*, ser. Computer Graphics Proceedings, Annual Conference Series. Los Angeles, California: ACM SIGGRAPH / Addison Wesley, August 1997, pp. 209–216.

[12] A. Greß and R. Klein, "Efficient representation and extraction of 2-manifold isosurfaces using kd-trees." *Graphical Models*, vol. 66, no. 6, pp. 370–397, 2004.

[13] G. M. Nielson, "Dual marching cubes," in *VIS '04: Proceedings of the conference on Visualization '04*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 489–496.

[14] J. Rossignac and P. Borrell, "Multi-resolution 3d approximation for rendering complex scenes," in *Modeling in Computer Graphics*, 1993, pp. 455–465.

[15] K.-L. Low and T. S. Tan, "Model simplification using vertex-clustering." in *SI3D*, 1997, pp. 75–82, 188.

[16] D. P. Luebke and C. Erikson, "View-dependent simplification of arbitrary polygonal environments." in *SIGGRAPH*, 1997, pp. 199–208.

[17] E. Shaffer and M. Garland, "Efficient adaptive simplification of massive meshes." in *IEEE Visualization*, 2001.

[18] P. Lindstrom, "Out-of-core simplification of large polygonal models," in *Proceedings of SIGGRAPH 2000*, ser. Computer Graphics Proceedings, Annual Conference Series. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, July 2000, pp. 259–262.

[19] M. Garland and E. Shaffer, "A multiphase approach to efficient surface simplification." in *IEEE Visualization*, 2002.

[20] D. Brodsky and B. Watson, "Model simplification through refinement." in *Graphics Interface*, 2000, pp. 221–228.

[21] T. Kanaya, Y. Teshima, K. ichi Kobori, and K. Nishio, "A topology-preserving polygonal simplification using vertex clustering." in *GRAPHITE*, 2005, pp. 117–120.

[22] T. Lewiner, L. Velho, H. Lopes, and V. Mello, "Simplicial isosurface compression." in *Vision, Modeling, and Visualization Conference*, 2004, pp. 299–306.

[23] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, and W. Wright, "Simplification envelopes," in *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press, 1996, pp. 119–128.

[24] D. Attali, D. Cohen-Steiner, and H. Edelsbrunner, "Extraction and simplification of iso-surfaces in tandem." in *Symposium on Geometry Processing*, 2005, pp. 139–148.

[25] T. Dey, H. Edelsbrunner, S. Guha, and D. Nekhayev, "Topology preserving edge contraction," *Publ. Inst. Math. (Beograd) (N.S.)*, vol. 6, pp. 23–45, 1999.

[26] G. M. Nielson and B. Hamann, "The asymptotic decider: resolving the ambiguity in marching cubes," in *VIS '91: Proceedings of the 2nd conference on Visualization '91*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1991, pp. 83–91.

[27] C. J. A. Delfinado and H. Edelsbrunner, "An incremental algorithm for betti numbers of simplicial complexes on the 3-sphere." *Computer Aided Geometric Design*, vol. 12, no. 7, pp. 771–784, 1995.

[28] T. Ju and T. Udeshi, "Intersection-free contouring on an octree grid," in *Pacific Graphics '06*, 2006.

[29] A. Szymczak and J. Vanderhyde, "Extraction of topologically simple isosurfaces from volume datasets," in *IEEE Visualization*, 2003, pp. 67–74.

[30] Z. J. Wood, H. Hoppe, M. Desbrun, and P. Schröder, "Removing excess topology from isosurfaces." *ACM Trans. Graph.*, vol. 23, no. 2, pp. 190–208, 2004.

APPENDIX I
PROOF OF PROPOSITION 1

Proof: We first show that the simplified contour contains only manifold edges. Let $\{v,w\}$ be an edge on the contour, and without loss of generality, let C_v be at an equal or finer level than C_w on the octree. The key is to observe that each polygon in the simplified contour containing $\{v,w\}$ corresponds to some polygon in the uniform contour S_v that intersects an edge of C_v. Due to condition (2), $\{v,w\}$ is contained in exactly two polygons.

We next show that each contour vertex is contained in a manifold neighborhood. The Euler characteristic of a connected, orientable 2-manifold S_v is related to the number of surface boundaries, $h(S_v)$, and number of surface handles (i.e., genus), $g(S_v)$, as:

$$\chi(S_v) = 2 - 2g(S_v) - h(S_v)$$

Since both g, h are non-negative integers, the only possible situation under which $\chi(S_v) = 1$ is when $g(S_v) = 0$ and $h(S_v) = 1$. In other words, S_v is topologically equivalent to an open disk with a single boundary cycle. The vertices and edges in this boundary cycle remains as a single connected component after vertex clustering, which forms the boundary of the 1-ring neighborhood of v. Since v is contained in only manifold edges, the 1-ring neighborhood of v is also topologically equivalent to a disk. $\square$

APPENDIX II
PROOF OF EQUATION 2

Proof: Consider $\overline{S_v}$ as the portion of S_v that lies within the faces of C_v, which consists of those polygons in S_v interior to C_v and all other polygons in S_v truncated by the six faces of C_v. We first show that $\chi(S_v) = \chi(\overline{S_v})$. Let $\partial(S_v)$ be the the set of vertices and edges the lie on the boundary of S_v. Since S_v is a 2-manifold, $\partial(S_v)$ forms closed cycles and therefore $V(\partial(S_v)) = E(\partial(S_v))$. Similarly, we have $V(\partial(\overline{S_v})) = E(\partial(\overline{S_v}))$. Since S_v and $\overline{S_v}$ share the same non-boundary vertices, edges and polygons, formula 1 yields $\chi(S_v) = \chi(\overline{S_v})$.

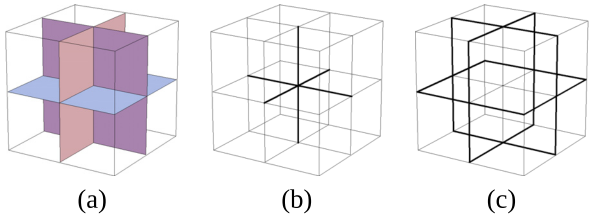

Fig. 13. Center planes of a cell (a), center lines of a cell (b), center lines of each cell face (c).

Next, as $\overline{S_v}$ is the union of all $\overline{S_{v_k}}$, we consider M_v as the set of vertices and edges that are contained in more than one $\overline{S_{v_k}}$. The key observation is that M_v lies on the 12 internal faces of the octree cell C_v (see Figure 13 (a)). Furthermore, we use M_v^c and M_v^f to denote respectively the set of vertices in M_v lying on the center lines of C_v (see Figure 13 (b)) and on the center lines of faces of C_v (see Figure 13 (c)). Observe that each vertex in M_v^c is contained in exactly 4 $\overline{S_{v_k}}$, whereas each other element of M_v is contained in exactly 2 $\overline{S_{v_k}}$. According to formula 1,

$$\chi(\overline{S_v}) = \sum_k \chi(\overline{S_{v_k}}) - (V(M_v) + 2V(M_v^c)) + E(M_v) \tag{3}$$

On the other hand, since each $\overline{S_{v_k}}$ is a 2-manifold, a vertex in M_v is contained in exactly 2 edges of M_v except for those vertices in M_v^c, each contained in 4 edges, and those in M_v^f, each contained in 1 edge. Hence we have,

$$\begin{aligned} 2E(M_v) = &\quad 2(V(M_v) - V(M_v^c) - V(M_v^f)) + 4V(M_v^c) + V(M_v^f) \\ = &\quad 2V(M_v) + 2V(M_v^c) - V(M_v^f) \end{aligned} \tag{4}$$

Substituting equation 4 into equation 3 yields

$$\chi(\overline{S_v}) = \sum_k \chi(\overline{S_{v_k}}) - \frac{2V(M_v^f) + 4V(M_v^c)}{4} \tag{5}$$

Equation 5 yields equation 2, because $\chi(S_v) = \chi(\overline{S_v})$, $\chi(S_{v_k}) = \chi(\overline{S_{v_k}})$, and each vertex in M_v^f and M_v^c contributes to one edge intersection in $d(S_{v_k})$ for 2 and 4 S_{v_k}. $\square$

Tao Ju graduated from Tsinghua University in 2000 with a BA degree in English and a BS degree in Computer Science. He received his Ph.D degree in Computer Science from Rice University in 2005. Tao is currently an assistant professor in the Department of Computer Science and Engineering at Washington University in St. Louis. His research interests are in the areas of mesh processing, visualization, geometric modeling, and biomedical applications.

Joe Warren, a Professor of Computer Science at Rice University, is one of the world's leading experts on subdivision. He has published numerous papers of this topic and its applications to computer graphics. These publications have appeared in such forums as SIGGRAPH, Transactions on Graphics, Computer-Aided Geometric Design and The Visual Computer. He has also organized and participated in a number of international workshops, short courses and minisymposia on the theory and practice of subdivision. Professor Warren's related areas of expertise include computer graphics, geometric modeling and visualization.

Scott Schaefer is an Assistant Professor in the Computer Science department at Texas A&M University. He graduated from Trinity University in 2000 with a B.S. degree in Computer Science and Mathematics, received an M.S. degree from Rice University in 2003 and a Ph.D. from Rice University in 2006. His research interests include Computer Graphics, Geometric Modeling and Scientific Visualization.